Code Review Guidelines

What is a code Review?

A code review is a detailed review of code and the end of the feature implementation or at logical intervals for validating the design and implementation of features/patches.

Why Reviews are important?

1. To spot and fix defects early in the process.
2. Better-shared understanding of the code base as team members learn from each other
3. Helps to maintain a level of consistency in design and implementation.
4. It is more affordable and can be more effective than testing process.
5. Helps to identify common defects across the team thus reducing rework.
6. Builds confidence of stakeholders about technical quality of the execution.
7. Uniformity in understanding will help interchangeability of team members in case of non-availability of any one of them.
8. In case third party reviews code doesn't get adverse comments.

Before we commit any code to source control, we review it for compliance with the list below:

- General Unit Testing
- Comment and Coding Conventions
- Error Handling
- Resource Leaks
- Thread Safety
- Control Structures
- Performance
- Functionality
- Security

Roles and Responsibilities

1. **Developer:** is the person who has written the code to be reviewed and has initiated the review request.
2. **Reviewer/s:** are the people who are going to review the code and report the findings to the developer.

Tips for the Developer:

1. **The primary reviewer is the author i.e. YOU.**

2. **Create a checklist for yourself of the things that the code reviews tend to focus on**. Some of this checklist should be easy to put together. It should follow the outline of the coding standards document. Because it's your checklist, you can focus on the thing that you struggle with and skip the things that you rarely, if ever, have a problem with. Run through your code with the checklist and fix whatever you find. Not only will you reduce the number of things that the team finds, you'll reduce the time to complete the code review meeting—and everyone will be happy to spend less time in the review.

3. **You are not your code.** Remember that the entire point of a review is to find problems, and problems will be found. Don't take it personally when one is uncovered.

4. **Understand and accept that you will make mistakes.** The point is to find them early, before they make it into production. Fortunately, except for the few of us developing rocket guidance software at JPL, mistakes are rarely fatal in our industry, so we can, and should, learn, laugh, and move on.

5. **No matter how much "karate" you know, someone else will always know more.** Such an individual can teach you some new moves if you ask. Seek and accept input from others, *especially* when you think it's not needed.

6. **Don't rewrite code without consultation.** There's a fine line between "fixing code" and "rewriting code." Know the difference, and pursue stylistic changes within the framework of a code review, not as a lone enforcer.

7. **The only constant in the world is change.** Be open to it and accept it with a smile. Look at each change to your requirements, platform, or tool as a new challenge, not as some serious inconvenience to be fought.

8. **Fight for what you believe, but gracefully accept defeat.** Understand that sometimes your ideas will be overruled. Even if you do turn out to be right, don't take revenge or say, "I told you so" more than a few times at most, and don't make your dearly departed idea a martyr or rallying cry.

9. **Don't be "the guy in the room."** Don't be the guy coding in the dark office emerging only to buy cola. The guy in the room is out of touch, out of sight, and out of control and has no place in an open, collaborative environment.

10. **Please note that Review meetings are <u>NOT</u>problem solving meetings.**

11. **Help to maintain the coding standards**. Offer to add to the coding standards for things discussed that aren't in the coding standards. One of the challenges that a developer has in an organization with combative code review practices is that they frequently don't know where the next problem will come from. If you document each issue into the coding standards, you can check for it with your checklist the next time you come up for code reviews. It also will help

cement the concept into your mind so that you're less likely to miss opportunities to use the feedback.

Tips for the Reviewer

1. **Critique code instead of people – be kind to the coder, not to the code.** As much as possible, make all of your comments positive and oriented to improving the code. Relate comments to local standards, program specs, increased performance, etc.
2. **Treat people who know less than you with respect, deference, and patience.** Nontechnical people who deal with developers on a regular basis almost universally hold the opinion that we are prima donnas at best and crybabies at worst. Don't reinforce this stereotype with anger and impatience.
3. **The only true authority stems from knowledge, not from position.** Knowledge engenders authority, and authority engenders respect – so if you want respect in an egoless environment, cultivate knowledge.

4. **Please note that Review meetings are NOT problem solving meetings.**

5. **Ask questions rather than make statements**. A statement is accusatory. "You didn't follow the standard here" is an attack—whether intentional or not. The question, "What was the reasoning behind the approached you used?" is seeking more information. Obviously, that question can't be said with a sarcastic or condescending tone; but, done correctly, it can often open the developer up to stating their thinking and then asking if there was a better way.

6. **Avoid the "Why" questions**. Although extremely difficult at times, avoiding the"Why" questions can substantially improve the mood. Just as a statement is accusatory—so is a why question. Most "Why" questions can be reworded to a question that doesn't include the word "Why" and the results can be dramatic. For example, "Why didn't you follow the standards here..." versus "What was the reasoning behind the deviation from the standards here..."

7. **Remember to praise**. The purposes of code reviews are not focused at telling developers how they can improve, and not necessarily that they did a good job. Human nature is such that we want and need to be acknowledged for our successes, not just shown our faults. Because development is necessarily a creative work that developers pour their soul into, it often can be close to their hearts. This makes the need for praise even more critical.

8. **Make sure you have good coding standards to reference**. Code reviews find their foundation in the coding standards of the organization. Coding standards are supposed to be the shared agreement that the developers have with one another to produce quality, maintainable code. If you're discussing an item that isn't in your coding standards, you have some work to do to get

the item in the coding standards. You should regularly ask yourself whether the item being discussed is in your coding standards.

9. **Remember that there is often more than one way to approach a solution**. Although the developer might have coded something differently from how you would have, it isn't necessarily wrong. The goal is quality, maintainable code. If it meets those goals and follows the coding standards, that's all you can ask for.

10. **You shouldn't rush through a code review- but also, you need to do it promptly.** Your coworkers are waiting for you.

Assign Severity to Review Finding

The severity to find issues with code should go as below. Reviewer must focus on issues with High severity first and then to Medium severity and then Low severity issues.

1. Naming Conventions and Coding style = Low
2. Control Structures and Logical issues = Medium or High
3. Redundant Code = High
4. Performance Issues =High
5. Security Issues = High
6. Scalability Issues= High
7. Functional Issues =High
8. Error Handling = High
9. Reusability = Medium

How to do code review

Before going for code review, please see the following checklist. This checklist should be stringently followed by the developer who is requesting the code review.

1. Does review Meeting should be scheduled prior to at least one day before the review requested? Y/N

2. Does meeting Request should contain following items. (All items are mandatory)

   - ✓ Objective of review (This can be decided in consultation of respective project manager/Tech lead/architect/reviewers)
   - ✓ Work Item/s of review (e.g. Use Case /User story number)
   - ✓ Unit testing for work item/s done? Y/N
   - ✓ Has the code committed to source control repository? Y/N

- ✓ Does Code complies and runs on another developer's machine without errors or warnings? Y/N

3. Reviewer gets request for review and s/he goes from work Items that has to be reviewed.
4. In Review meeting, if reviewer has any findings beforehand, then s/he shares it with the developers.
5. Then, developer explains the functional requirement in brief and also gives walk through of code.
6. Reviewer document the review findings along with some explanation and action plan for developer.
7. After the meeting, developer corrects the findings and eliminates the issues.
8. Developer again unit test the code.
9. Developer conveys the reviewer that all issues have been resolved and waits for the acknowledgement email from the reviewer.
10. Reviewer re-checks the code modifications in source control repository and sends the acknowledgement email to developer stating the work items and their status whether they are closed or still open. If items are open then developer has to follow all the steps from step 6.

References:

1. http://www.codinghorror.com/blog/2006/05/the-ten-commandments-of-egoless-programming.html
2. http://www.developer.com/java/other/article.php/3579756
3. http://www.smartbear.com/docs/BestPracticesForPeerCodeReview.pdf